IACTSIM: A CUDA-ACCELERATED PYTHON SIMULATION FRAMEWORK FOR IACTS

Davide Mollica[1], Ciro Bigongiari[2,3], Antonino D'Aì[1], Fabio Pintore[1]

[1]INAF, Istituto di Astrofisica Spaziale e Fisica Cosmica, Via Ugo la Malfa 153, I-90146 Palermo, Italy;
[2]INAF-Osservatorio Astronomico di Roma, Via Frascati 33, 00078 Monte Porzio Catone (Roma), Italy;
[3]ASI – Space Science Data Center, Via del Politecnico s.n.c., 00133 Roma, Italy

iactsim

Introduction

Analyzing data from Imaging Atmospheric Cherenkov Telescopes (IACTs) requires large-scale Monte Carlo simulations of air showers and the detailed simulation of telescope optics and Cherenkov camera. Within the simulation of the telescope response, optical ray-tracing and camera electronics are the most time-consuming parts. Fortunately, these tasks involve many independent calculations (photon propagation and pixel response) that can be run concurrently. This makes them ideal candidates for acceleration using Graphics Processing Units (GPUs).

We have developed `iactsim`, a Python simulation framework, using CUDA to parallelize these specific tasks. This framework is designed as a user-friendly and flexible set of tools to support IACT performance evaluation, instrument design and data analysis.

`iactxx`

`iactxx` is a `pybind11`-based C++ module designed to solve I/O bottlenecks when processing compressed files written with the CORSIKA7 IACT extension. It enables parallel reading, decompression e processing of photon bunches into a GPU-optimal structure-of-arrays format, enabling efficient data transfer to GPU memory.

ASTRI use case

We have validated the framework capabilities by simulating the ASTRI dual-mirror optical system (see Fig. 2) and its SiPM-based camera (see Fig. 3) [Scuderi et al., 2022, JHEAP, 35, 52]. Preliminary results have been obtained using a configuration based on ASTRI-1 measurements. The simulated optical system includes the 18 primary mirror hexagonal segments, the secondary mirror, the camera protective window and all main shadowing elements. The camera topological trigger logic has been implemented, as well as the signal sampling method based on a peak-detection circuit.

Optics

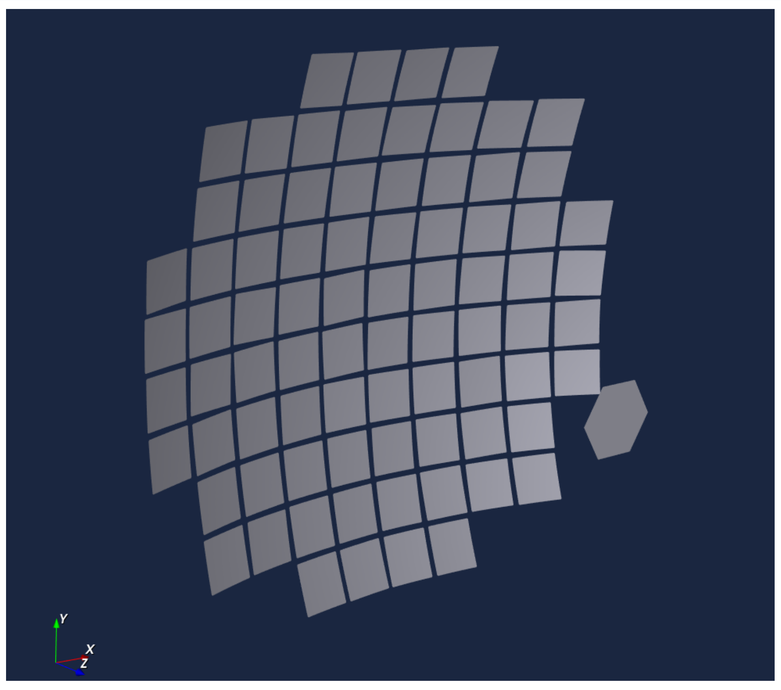

Fig. 1: An example single-mirror system with a segmented mirror and an hexagonal flat focal surface. The non-sequential ray-tracing algorithm traces the rays reflecting off each of these individual mirrors onto the focal surface.

The optical framework core consists of CUDA kernels that perform a non-sequential ray-tracing. These kernels are managed by a `IACT` class, which handles kernel calls and GPU data transfers. An optical system can be built from any number of surfaces, represented by Python classes that define their properties like shape, position and tilt. This approach provides significant flexibility, allowing the creation of systems ranging from simple toy-models (Fig. 1) to realistic instrument models (Fig. 2). Furthermore, these surfaces can act as interfaces between two materials, enabling the simulation of refractive elements.

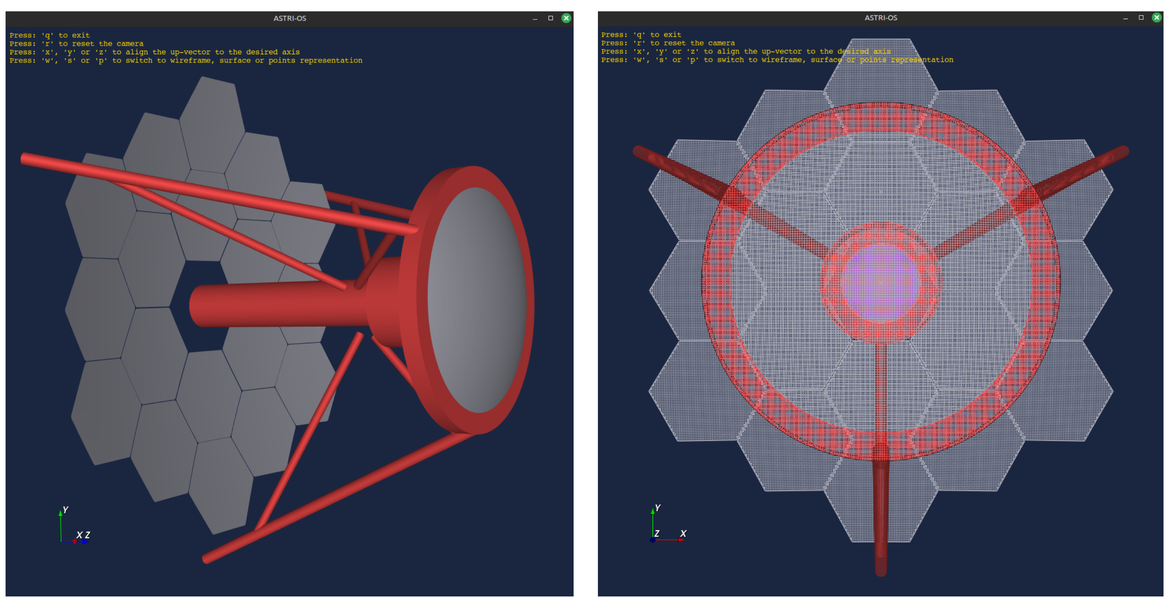

Fig. 2: Left) Global view of the implemented ASTRI telescope geometry showing mirrors (grey) and shadowing elements (red). Right) Front wireframe view showing the camera window (blue).

Cherenkov cameras

The behavior of the camera electronics is defined through a hierarchical class structure

- `CherenkovCamera` is the abstract base class that defines the camera general behaviour:
  - it is composed of n pixels and m channels (trigger, sampling or auxiliary);
  - methods for handling channels signal, trigger logic and post-trigger signal processing must be implemented to create a custom camera.
- `CherenkovSiPMCamera` is a derived abstract class for SiPM cameras:
  - it defines SiPM-specific signal computations;
  - only the trigger logic and the sampling method must be defined to create a custom SiPM camera.

To facilitate implementation, the framework includes CUDA kernels to simulate components like ideal discriminators and analog-to-digital converters.

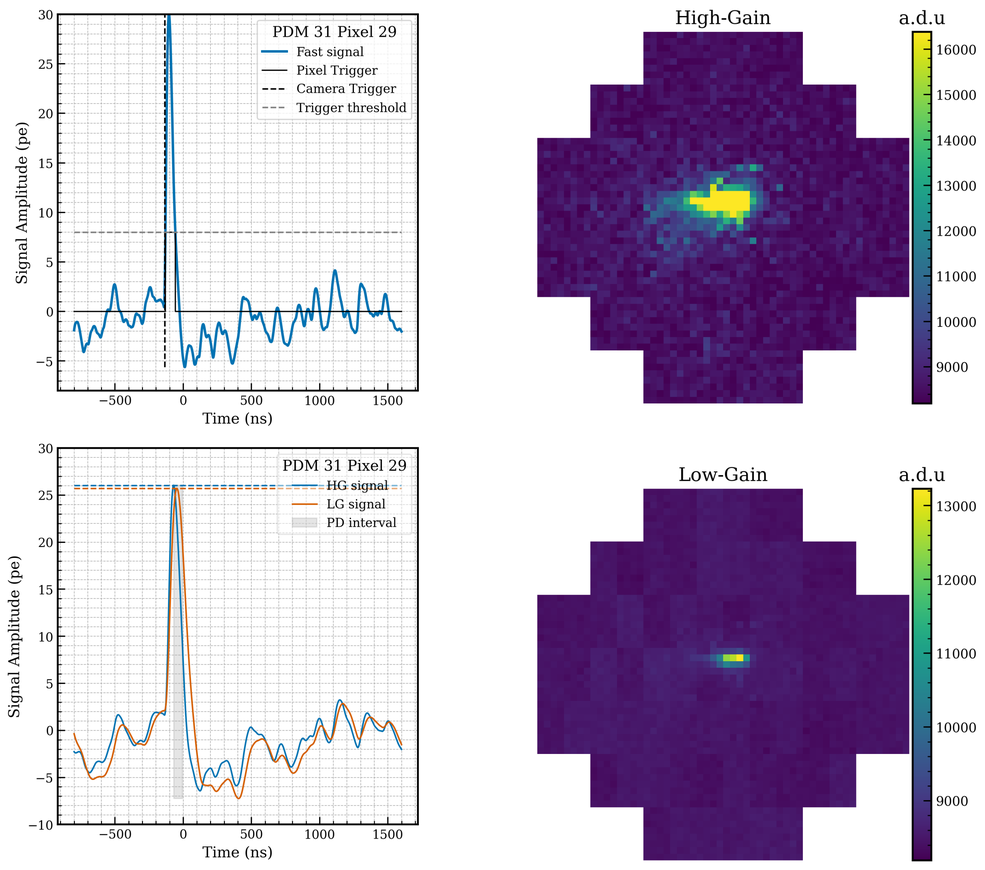

Fig. 3: Left) Example of ASTRI trigger (top) and sampling channels (bottom) simulation of a Cherenkov pulse with night sky-background noise. The channel discriminator output and the peak-detection interval are shown. Right) Simulated ASTRI telescope detection of a proton shower in the High-Gain (top) and Low-Gain (bottom) sampling channels.

Conclusion

We have validated the framework flexibility by simulating a variety of ASTRI calibration procedures (an example is shown in Fig. 4). The preliminary results are promising and the simulator is currently being configured with field measurements to match the actual ASTRI-1 camera and optics response. In addition, we are also carrying out a performance comparison with the currently implemented `sim_telarray`-based ASTRI simulator. Preliminary results suggest a speed-up of two orders of magnitude. However, we found that the reading of CORSIKA output files can be a significant performance bottleneck.

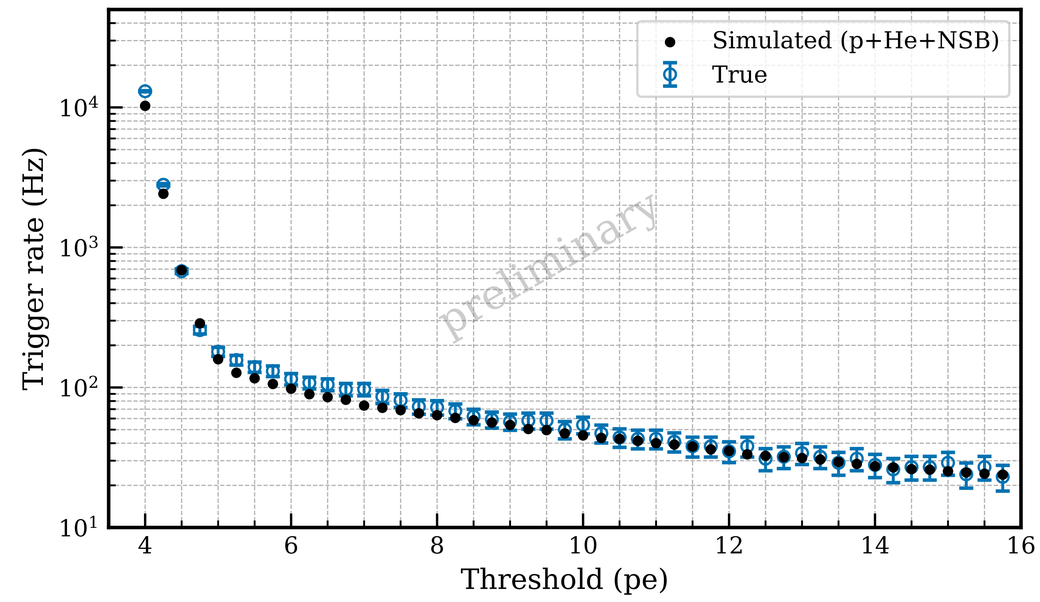

Fig. 4: Comparison between measured and simulated ASTRI-1 threshold scan. The night-sky background rate was simulated on a 2 s time window. The shower detection rate was simulated with 10^6 showers per species (proton and helium), with a -1.5 energy slope and the results re-weighted from DAMPE spectra.

Acknowledgments

This work was conducted in the context of the ASTRI Project. We gratefully acknowledge support from the people, agencies, and organisations listed here: http://www.astri.inaf.it/en/library/. This paper went through the internal ASTRI review process.